

Сокращаем дифы, снижаем каплинг, продлеваем жизнь

Лещёв Иван



Приветствие



Лещёв Иван

- php-боярин
- санитар философии
- а ещё я работаю ВКонтакте

https://vk.com/ileshchev

https://t.me/Ivan_Leshchov

Очевидные мысли

- Кажется, мы всё ещё не умеем писать хороший код
- Кажется, мы не знаем, что это такое
- Определённо, хороший код! = плохой код
- Определённо, если не трогать код, то как будто и наплевать на него
- А вот если начать с ним работать…

Код надо писать так,

как если поддерживать
его будет эмоционально-
неустойчивый человек
с мачете, который знает,
где вы живёте.



Плохой код

Классический пример

Новая фича

/Users/i.leshchev/PhpstormProjects/skeleton/src 6 files
- Controller 1 file
  - FooController.php
- DTO 1 file
  - SomeFoo.php
- Repository 2 files
  - FooRepository.php
  - FooRepositoryInterface.php
- Service 2 files
  - FooService.php
  - FooServiceInterface.php

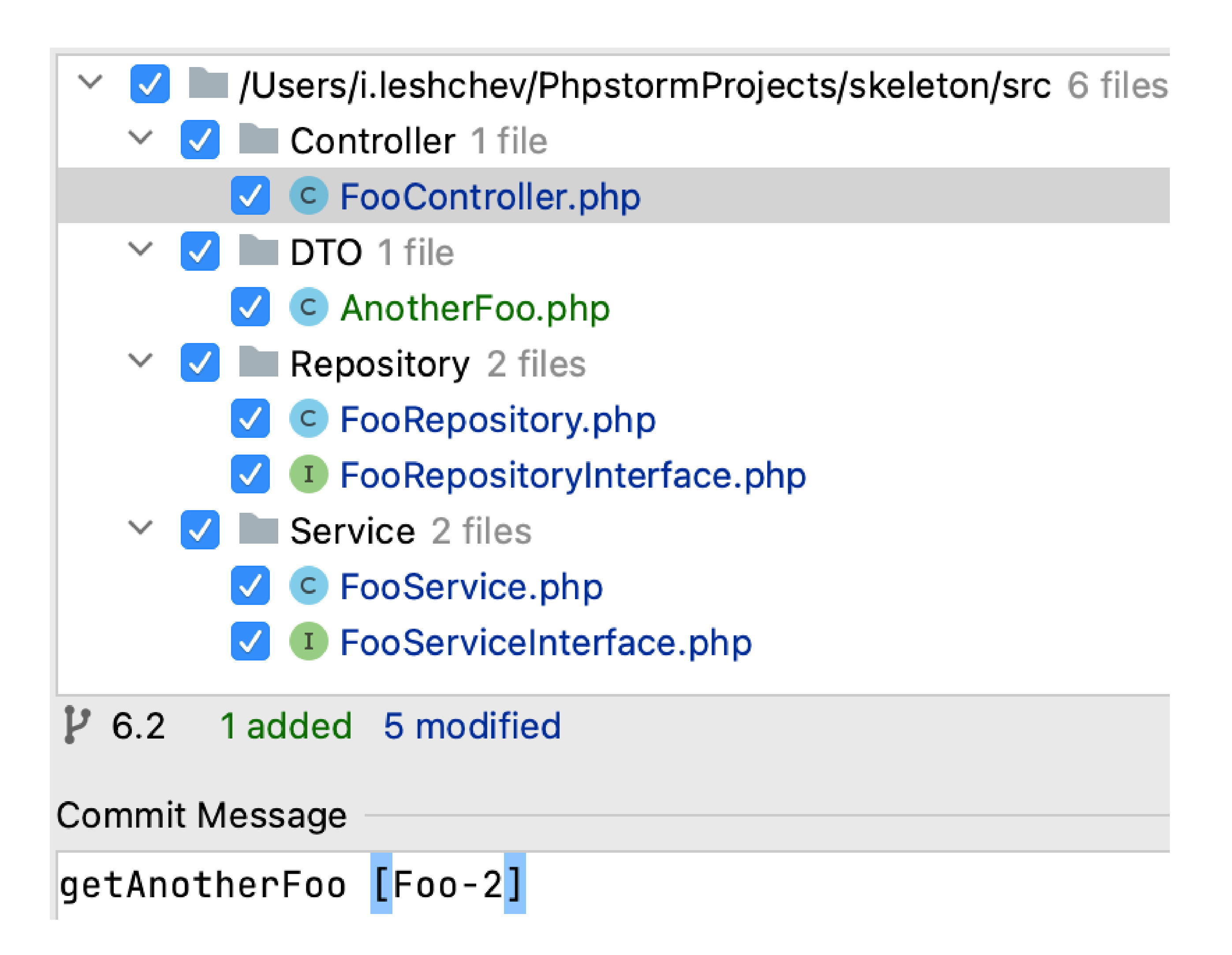
6.2 6 added

Commit Message

```
getSomeFoo [Foo-1]
```

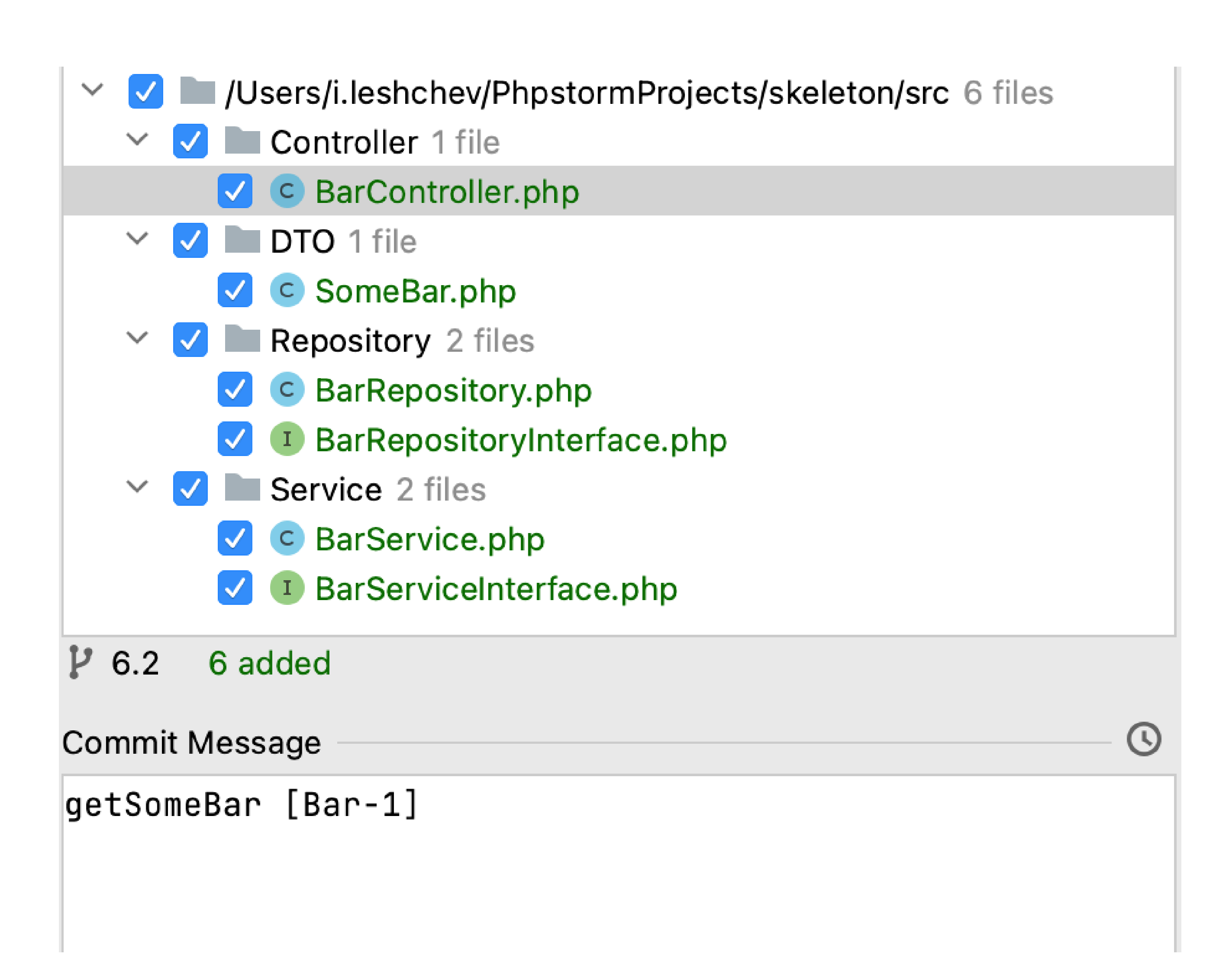
Плохой код

Добавили немножк



Плохой код

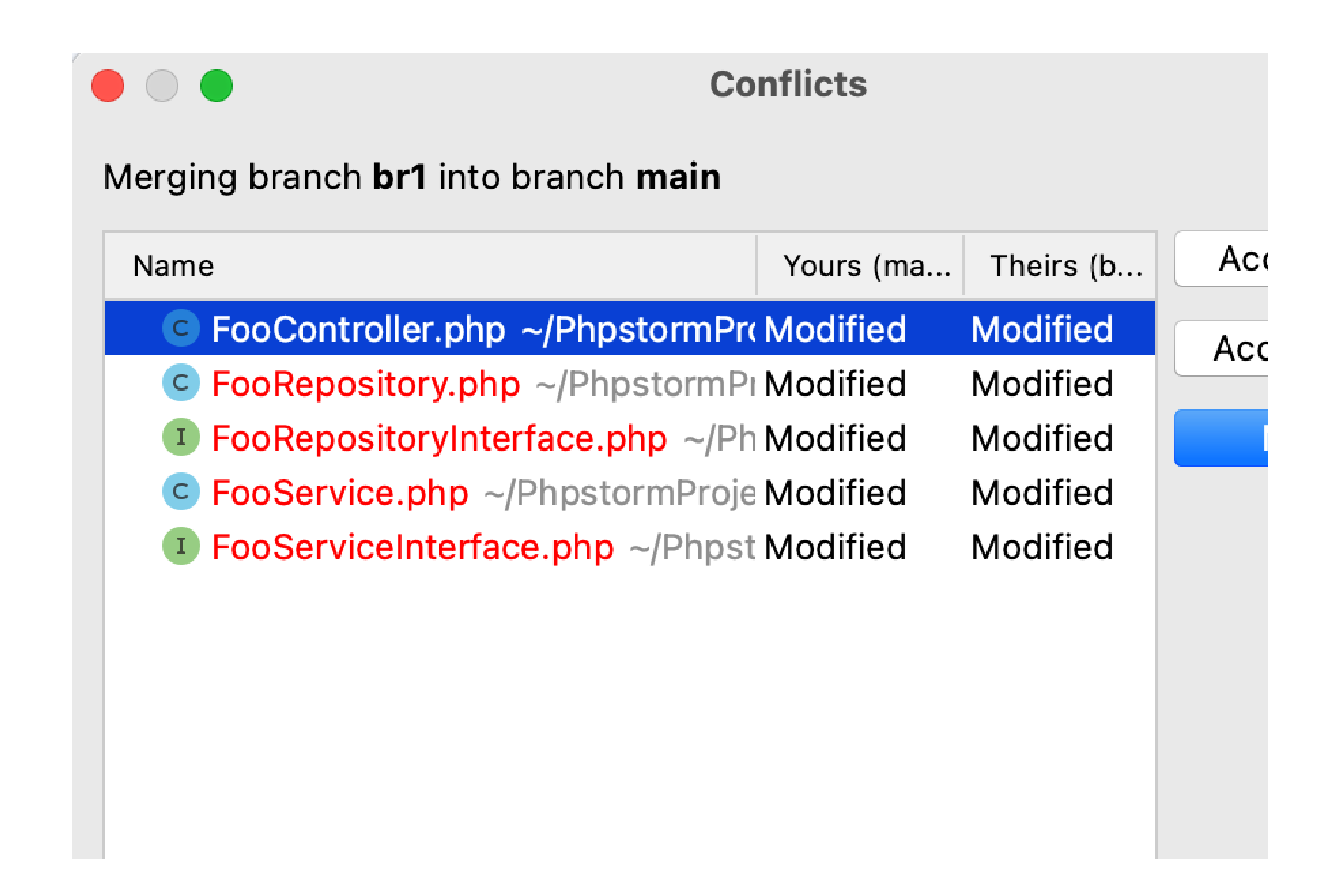
Другая фича



Плохой код

Параллельная разработка.

Не самый плохой случай!



Плохой код

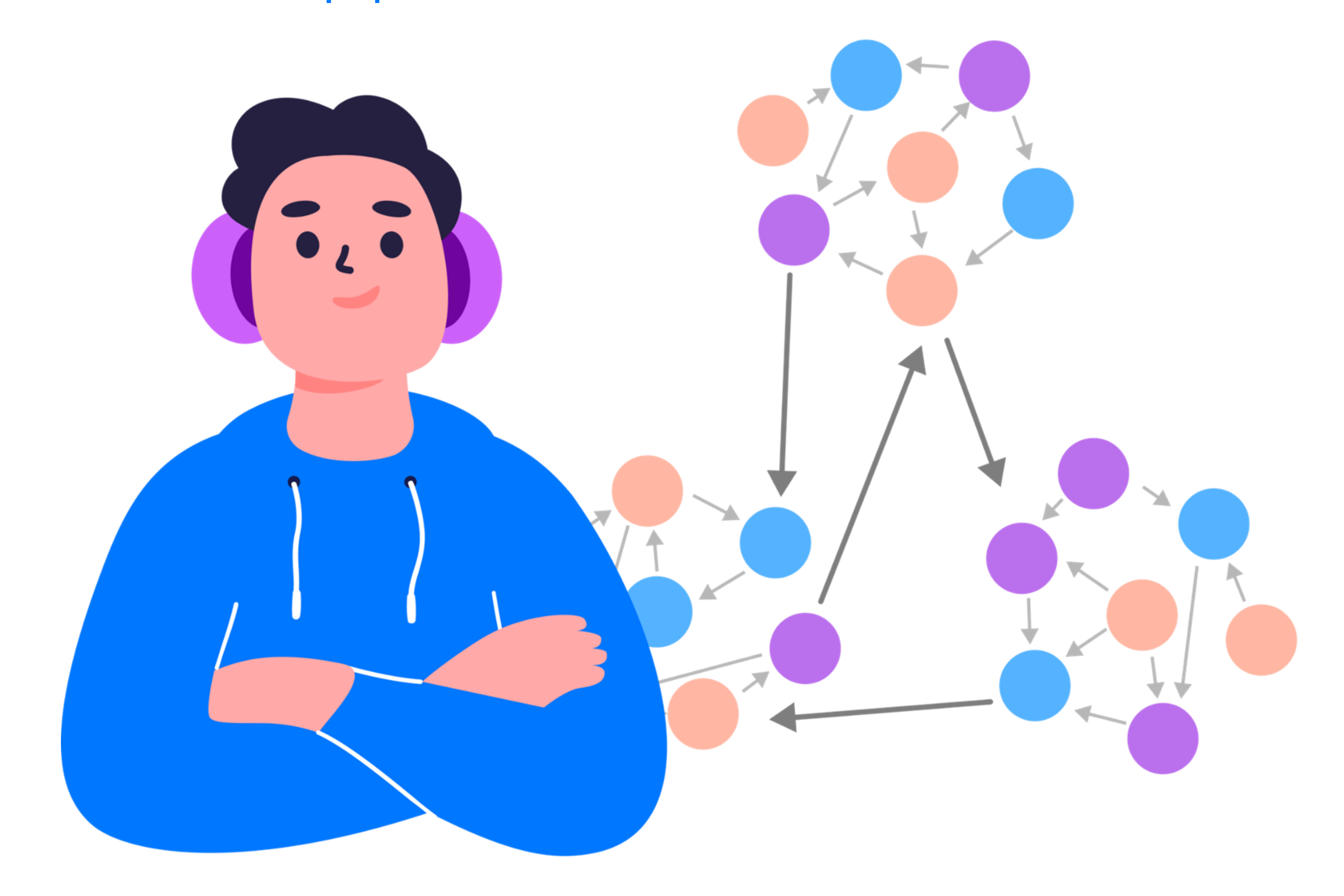
Конфликтов слияния нет, но лучше бы они были.

- Потеряли use
- Задублировали use

Плохой код

NDA

Плохой код



Плохой код

```
типа функция(ввод)
  получили данные а
  отвалидированили а
  получили данные б
  отвалидировали б // где-то тут положили б в статическое поле статического класса
  отвалидировали а и б
  сконвертировали данные в с
  отвалидировали а и с
...
  обработали с // где-то тут взяли из статического поля статического класса
  вызвали метод, который вызывает метод, который что-то делает с б, но это не точно
  вывод
```

Плохой код

- Даже маленькие правки затрагивают много файлов
- И разных директорий
- Непонятная структура
- Конфликты

- Сложно понимать
- Сложно изменять
- Сложно удалять

Хороший ход



Хороший ход

- Декомпозиция
- Сохранение единой ответственности

Хороший ход

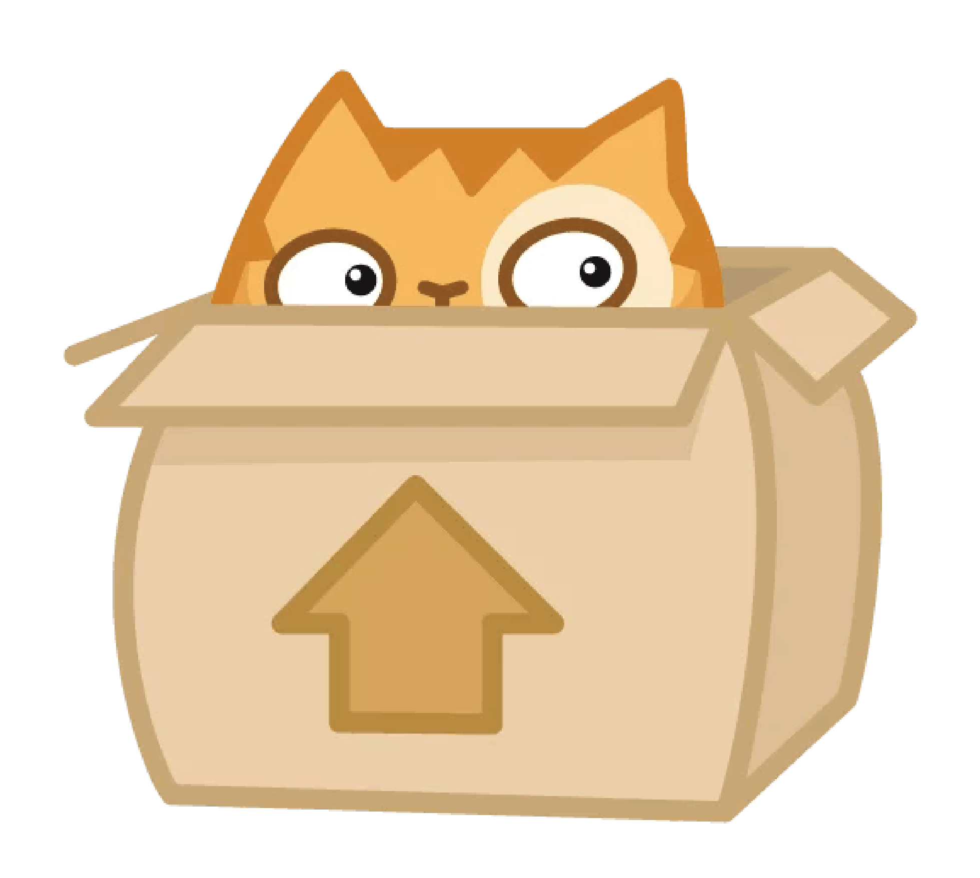
- С декомпозицией понятно, а что с композицией?
- Данные + код = объект

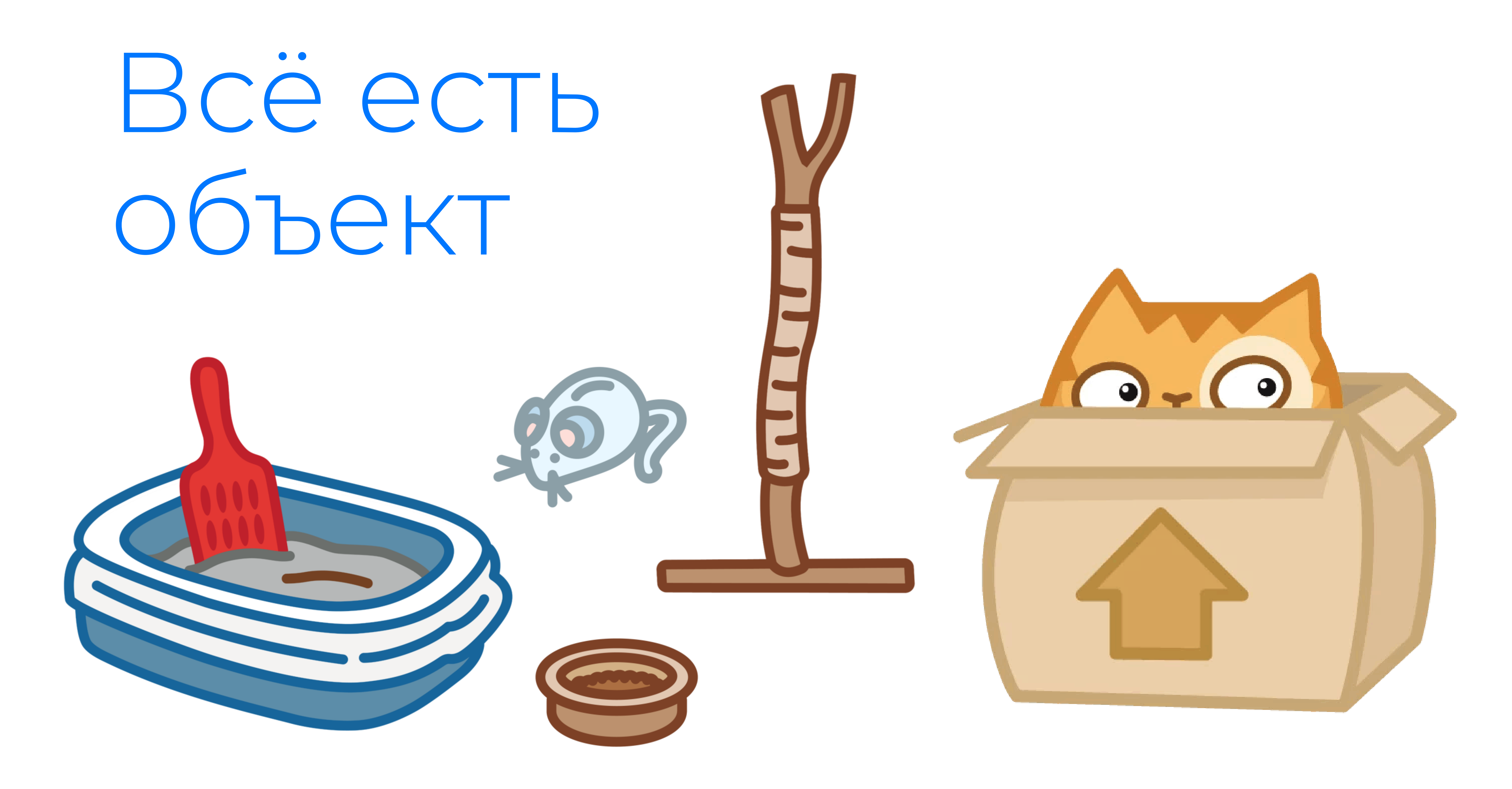
ООП

Принципы Алана Кея

- Всё есть объект
- Объекты общаются сообщениями

Всё есть объект

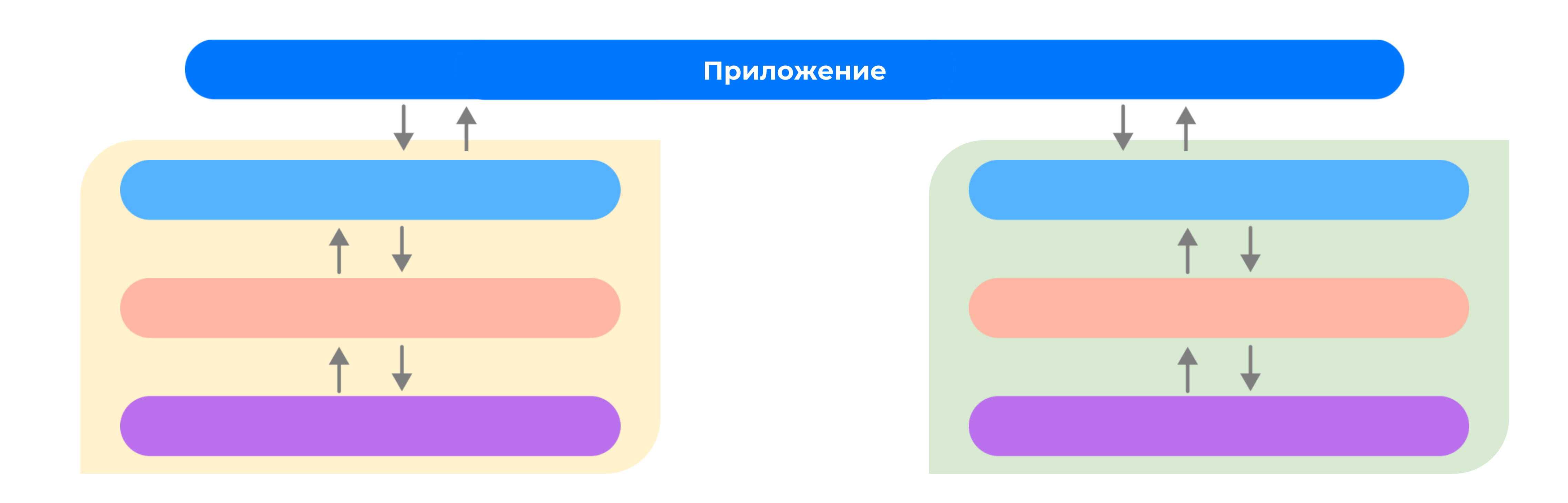



Всё есть объект

Фича — это объект

- Фича попадает под квантор «всё»
- Фича — композиция данных и обрабатывающего их кода
- Фича общается сообщениями настоящим, а не вырожденным образом

Фича — это объект



Фича — это объект

- Группировка по фиче — проще понимать, проще удалять, проще извлекать в микросервис
- Инкапсуляция внутри фичи
- CQRS — это база, не общаемся сущностями, а только сообщениями
- Сообщения команды, сообщения уведомления, сообщения статусы

Инкапсуляция

```php
    #[ORM\Column(type: 'string', length: 255)]
    private string $title;

    public function getTitle(): string
    {
        return $this->title;
    }

    public function setTitle(string $title): void
    {
        $this->title = $title;
    }
}
```

```php
    #[ORM\Column(type: 'string', length: 255)]
    public string $title;
}
```

Инкапсуляция

- Бизнес сущность полностью скрыта внутри фичи
- Механизма приватов и пабликов не хватает, их никогда не хватало
- Нужны статические анализаторы, всегда были нужны
- Можно избавиться от бесполезных приватов в сущности и тестов на примитивные сеттеры/геттеры

Обработка команд

- Отправка и обработка сообщения команды — единственный способ взаимодействовать на бизнес сущность

- Строго по сценарию — валидация, блокировка, действие, сохранение, разблокировка, уведомления

- Одна команда — один обработчик

Обработка команд

- Меньше кода
- Лучше композиция
- Проще ревью

```php
public function __invoke(Command $command) {
    $this->validator->validate($command);
    $this->locker->lock($command);
    $this->handle($command);
    $this->locker->unlock($command);
}
//.....
protected function handle(Command $command)
    $foo = $this->repository->getSomeFoo();
    //....
    $this->repository->flush();
}
```

Жизнь сообщения

Или ещё проще.

```
public function __invoke(Command $command) {
  $foo = $this->repository->getSomeFoo();
  //....
}
```

Жизнь сообщения

В большинстве случаев объекты сообщения достаточно хороши, как контракты, чтобы просто делать вот так.

```
class SomeReadDTO {
  public function __construct(
    public int $id,
    public string $title,
    public string $body,
    public int $timestamp,
  ) {
  }
}

$data = new SomeReadDTO(...$row);

$json = json_encode($data);
```

Жизнь сообщения

Или так.

```
#[Route(path: '/{id}', name: 'update',
  requirements: ['id' => "\d+"], methods: ['PUT'])]
public function update(int $id, Request $request): .
{
  $data = ['id' => $id] + $request->toArray();
  $this->bus->dispatch(new Command(...$data));

  return $this->json(['success' => true]);
}
```

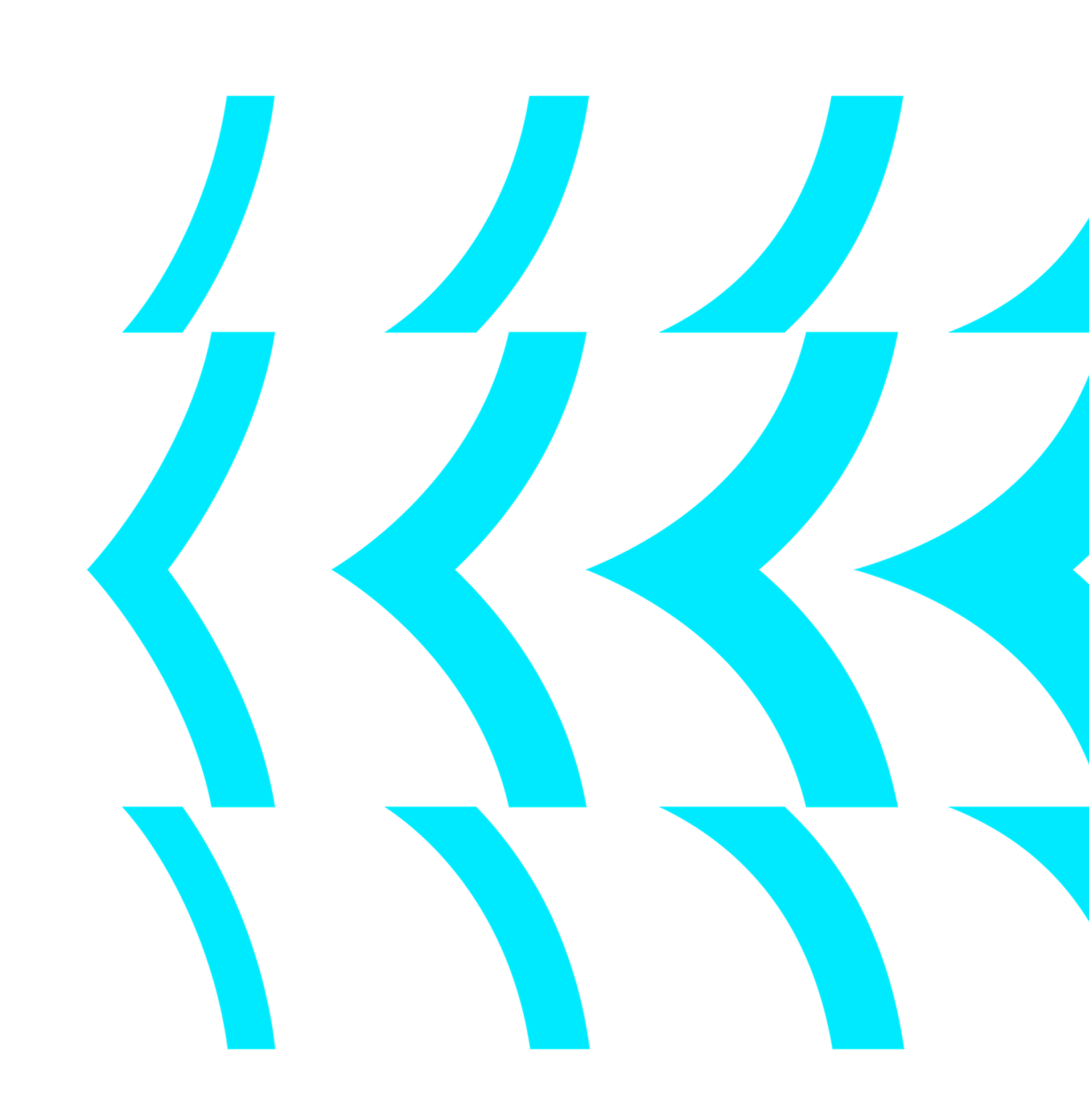
Итого

- Фичи – полноценные объекты с инкапсуляцией и интерфейсами-сценариями

- Фичи общаются сообщениями, которые суть контракты и определяют границы декомпозиции

- Инкапсуляция и сокрытие внутри фичи требует свежего взгляда и отдельных инструментов

Закон конвея



Выполняем

Фича
Сценарий
Сообщение
Аспект
Цикл разработки

Фича
Сценарий
Сообщение
Аспект
Цикл разработки

Не выполняем

Фича

Сценарий

Сообщение

Аспект

Цикл разработки



Цикл разработки

- Пишем тесты
- (Не) меняем реализацию
- Собираем статистику и добавляем метрики
- A/B тесты, фича тогглы и вот это вот всё

Цикл разработки

```
final class Service
{
    public function do(){ }
    public function doFacadeUno(){ }
    public function doFacadeDuos(){ }
    private function internalUno(){ }
    private function internalDuos(){ }
    private function internalTres(){ }
}
```

Цикл разработки

```
final class Service
{
    public function do(){ }
    public function doFacadeUno(){ }
    public function doFacadeDuos(){ }
    private function internalUno(){ }
    private function internalDuos(){ }
    private function internalTres(){ }
}
```

влезть сюда

влезть сюда

влезть сюда

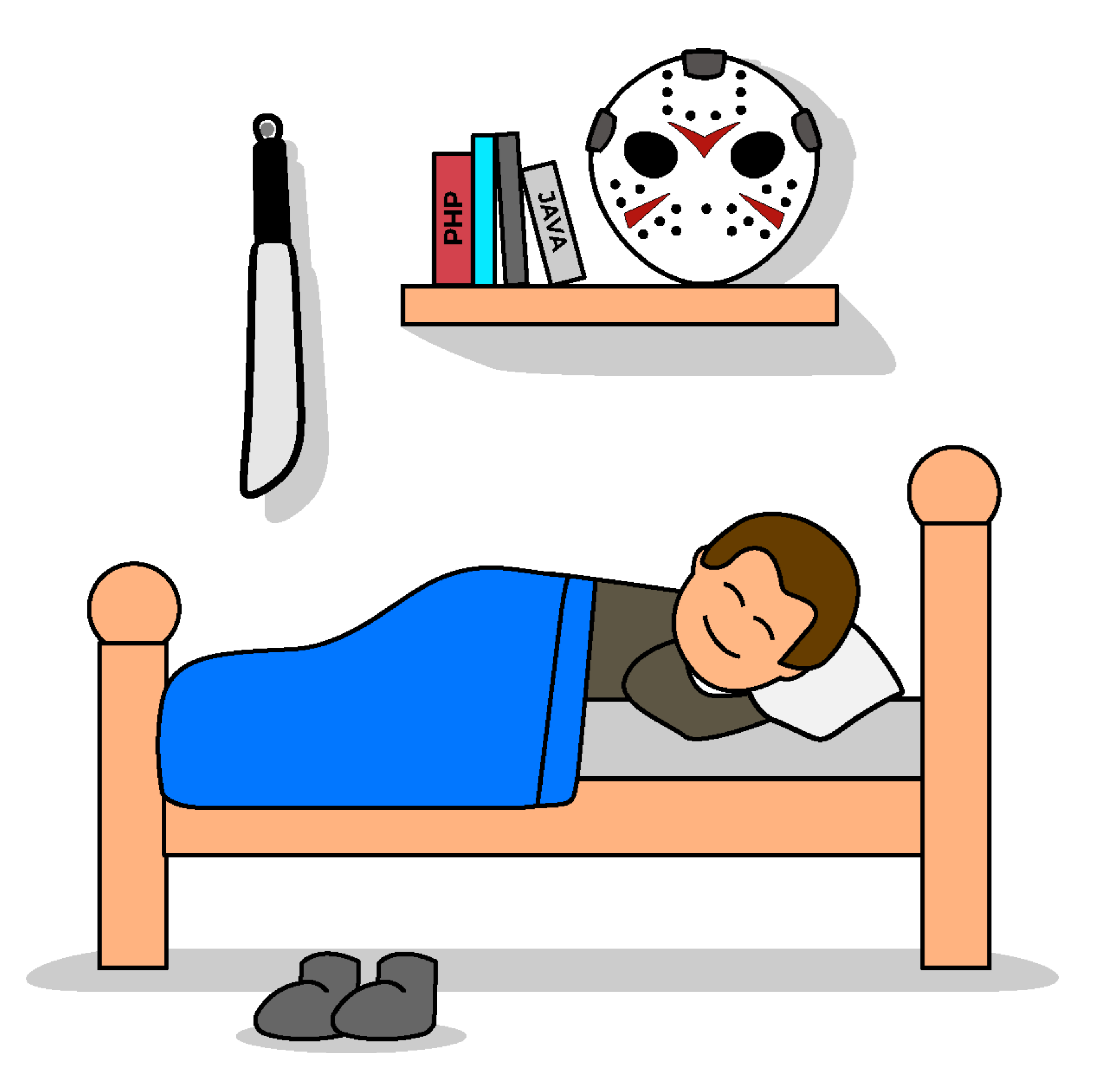
Цикл разработки

```
final class Service
{
    public function do(){ }
    public function doFacadeUno(){ }
    public function doFacadeDuos(){ }
    private function internalUno(){ }
    private function internalDuos(){ }
    private function internalTres(){ }
}
```

Цикл разработки

```
class Service
{
    public function do(){ }
    public function doFacadeUno(){ }
    public function doFacadeDuos(){ }
    protected function internalUno(){ }
    protected function internalDuos(){ }
    protected function internalTres(){ }
}
```

Всем хорошего кода



Оценивайте доклад и задавайте вопросы!



